REMARKS

Claims 14 and 15 stand rejected under 35 U.S.C. §112, second paragraph, as being indefinite. Claims 1 - 7 and 14 - 15 stand rejected under 35 U.S.C. §101 as being directed to non-statutory subject matter. Claims 1 - 3 and 5 - 7 stand rejected under 35 U.S.C. §102(e) as being anticipated by Lok (US 2002/0049530 A1). Claim 4 stands rejected under 35 U.S.C. §103(a) as being unpatentable over Lok in view of Polk et al (US 5,634,002). Claims 8, 10 and 11 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Lok in view of Berczik et al (US 2002/0054104 A1). Claims 9 and 12 - 15 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Lok in view of Berczik et al (US 2002/0054104 A1). The Examiner's thorough review is most appreciated, and has facilitated the amendments herein above. In view of the above amendment and following remarks, the Examiner is respectfully requested to reconsider the outstanding objections and rejections and allow the present application to issue.

With regard to the rejections under 35 U.S.C. §112, second paragraph, and 35 U.S.C. §101, claims 1 - 7 have been canceled herein, and claims 14 and 15 rewritten to remove the reference to "programmerator" and to incorporate the statutory process subject matter similar to that found in original claim 8.

With regard to the rejections under 35 U.S.C. §102(e) and 35 U.S.C. §103(a), the identification and recognition of the relevance of the Lok et al reference is appreciated. Neither applicants nor their representative would have recognized the relevance had the Examiner not both located the document, and then taken the time to carefully document the relevant features. By way of the present amendment, the claims now more clearly recite what the applicants regard as the invention.

Stated in a general way, one of the very important objects of the present invention is to avoid disruption of the computer System-Under-Test (290, Fig. 2). The aim is to not risk further unknown interaction between different system software, driver version differences, and the myriad of other like problems which have extremely complicated the prior art processes of

software testing and support.

More specifically, each time different software is loaded on a system, whether for diverse purpose or not, there is an increased likelihood of compatibility problems with the existing software. For example, loading Microsoft Excel, Corel WordPerfect, and Adobe Acrobat may lead to compatibility-related program failures, where only loading Corel WordPerfect and Microsoft Excel may be fine. The underlying problem is that different programs were created using particular drivers and OS versions. Each program, in order to ensure proper operation, may require a particular version of drivers and OS components. Software writers, when they realize that different versions may be an issue, may resort to installing particular drivers, dynamic link libraries (DLLs) or even OS DLLs or other components. Unfortunately, one program may require one version of these associated components, while another program may require a different version. In such instances, one or the other of the programs will fail, and so the two programs are not compatible. The chance of this happening increases with each different program that is installed, including different BIOS versions that are associated with the underlying hardware. When these variables are all figured in, it is understandable then that compatibility issues are so common and identification of bugs within programs so difficult to ferret out in a "fully loaded" computer.

The present invention seeks to avoid further loading a computer system with potentially incompatible software. The inventors have recognized that this problem is greatly worsened by software which operates at lower levels. In other words, a computer user will get one set of results or behaviors when operating the computer with the basic set of programs installed. If test software interferes or alters the way the computer operates, then a second set of results will be produced which are different from what the user would get without the test software. Due solely to the presence of the test software, performance results may thereby undesirably be corrupted.

In order to avoid such corruption, the present test software is designed to have as minimal a footprint on the System-Under-Test (SUT) 290 as possible. Most preferably, this is achieved

through a monitoring occurring only at the highest user level of the test OS, the Graphical User Interface (GUI). Mouse clicks are passed into the OS directly through software rather than through mouse port, but both act upon the OS GUI at the same software code, and thereby avoid software differences that will occur between accessing the mouse click at the OS GUI versus simulating a mouse click through an API interface. In the case of the API interface, additional and different code must be executed to execute the mouse click, and this additional and different code may ultimately result in different results than when executed directly through the GUI. Likewise, rather than rely upon two sets of software on separate computers to behave the same way, the present invention transmits pixel information back to the local computer from the remote computer. The pixels are then evaluated by the present invention, rather than a lower-level API call.

These differences may seem subtle to the Examiner or to a casual observer. However, they are of paramount importance in the world of computer monitoring, testing and control. To point out the limitations of the prior art, the Examiner is asked to consider the primary reference in the outstanding office action. The Lok et al patent relies upon two ***different*** API programs. One is the remote-capable user interface toolkit 108. The other is the baseline user interface toolkit 110. The API programs themselves are necessarily different, and these differences can result in unexpected and different behavior between the two systems. These two toolkits are then installed on two different computers. The two computers, referred to as server 102 and client 104, will undoubtedly have two different sets of programs installed thereon, with different hardware, different BIOS, and different drivers. Even the operating system may be different, or have a different version. Consequently, there is no certainty that an API call made at client 104 will generate the same result at server 102 as is generated at client 104. When a difference in result does arise, where should a developer or support person look? The difference may be in any of the aforementioned differences, from the API programs themselves to the OS, BIOS, hardware, and drivers, making the identification of the source of the different result impossible to

practically locate.

Furthermore, according to the Lok et al approach, each program must be capable of or be written to interact with a single specific API. Those familiar with software development will recognize that this also introduces many undesirable limitations. The most obvious is the requirement for each programmer to use the same API to create objects such as the graphical objects on the screen. Lok et al consider this to be a benefit, where on page 4 in paragraph 0040 they state: "End users also benefit, since most of the applications they run on a particular operating system will have roughly the same 'look and feel' because the applications are all built out of components from the same user interface toolkit." Without using the same API toolkits, the program will not use the same API call on Lok et al's client 104 to create a command button as is required to create the command button on Lok et al's server 102, and Lok et al will fail.

A quick review of the status of programming languages reveals that there are many different programming languages in commercial use today, and with each language comes a different set of tools to create system objects. A programmer may choose from C, C++, C#, BASIC, Fortran, APL, Java, Lisp, Pascal, Logo, COBOL, Perl, and many other programming languages. According to industry publications, even the most popular language, Java, is only used for some 20% of software development. Further compounding the issue, these languages are frequently implemented by different software companies that produce "IDEs", or Interactive Development Environments. Within each IDE from each different vendor, different techniques are or may be used to call operating system components. Even Java can differ, since it was originally developed by Sun Microsystems but adapted and altered by Microsoft. That means that even among Java developers, there will likely be two different sets of components, looks and function. Further compounding the problem, within each operating system (OS) version there are different techniques which are required or necessary to effect objects or events. So, not only must the numbers of programming languages and different vendors of each be considered, so must the numbers of OS choices. Quite frankly, it is unreasonable or impossible to suggest that a

single API could or would reasonably be used by all programs. Yet, even were it possible to make all developers use a single API, there will still remain the same problems that exist today with compatibility between the various components and programs.

To summarize then, the Lok et al invention provides teachings for a very controlled environment to generally successfully implement some of the basic concepts of the present invention, including remote access, viewing and control of a System-Under-Test (SUT). However, owing to the way Lok et al implement this capability through the API, which is very different from GUI access of the present invention, the Lok et al system can only be used in a carefully controlled environment. For exemplary purposes, in a tightly controlled corporate environment, where all computers are purchased having the same hardware, OS and loaded software, then an IT person might successfully use the Lok et al invention to control and potentially update or otherwise interact with each of these individual computers. In the event there were an interaction problem, the IT person would immediately be able to identify the source of the problem, since either the IT server has the same problem with all clients, or one client is different from the rest. In some corporations, this has been the approach of the IT department.

However, for a typical corporation having many different departments, this approach is highly undesirable. IT must rule with an iron fist for this approach to work, and keep all systems both identical and compatible with the API. So how does the engineering department purchase and install a new CAD package? Recognize that the CAD package was probably not written to call or use the API presented by Lok et al. Next, we have the accounting department who wants to implement a new accounting package. Sorry says the IT department, the new accounting package is not compatible with the API. And so it goes. Instead of each department being able to purchase the best software for its needs, the IT department must determine the selection based upon compatibility with a single API. From the above figures regarding IDE languages, the Examiner will recognize that even an extremely popular API would likely only be incorporated

into a few percent of all programs. As a result, the Lok et al system which at first sounds practical is little more than a continuation of the problem that already exists, which is that there are many different programs, and maintaining compatibility among all of them is impossible.

Instead, the present inventors have not tried to force any type of compatibility. The SUT 290 of the present invention is free to operate within its own unique OS, hardware, BIOS, and the like without the need for any additional API layer to generate screen objects or events. The screen objects are created entirely by whatever software may be running on the SUT, and events are created directly by the user or through remote system calls into the OS GUI at the same software level as accessed by the user when creating user events. This is described, for example, within the present specification on page 9 in lines 3 - 6, which state: "These commands will most desirably replicate the functions at the system-under-test 290 as though they were, in fact, executed directly upon that system. Preferably, such commands will include keyboard commands and mouse commands, though it will be understood that any form of user input may be emulated. Consequently, touch screen monitors, graphics pads or tablets, and any other type of primary or peripheral input device may be emulated as required and designed for." On page 10 in line 1: "Operation of the present invention depends upon the graphical user interface." On page 10 in lines 9 - 12: "Instead the user of interactive development environment 100 has complete control over any of the ***user actions*** that may be relayed to the system-under-test 290, such as providing typed text, commands, movement of the mouse, and so forth." (Emphasis added) From page 10, lines 19 - 21, "Since the present invention is designed to control graphical user interfaces, several commands are contemplated herein, but once again not considered to be solely limiting or restricted thereto." From the foregoing and the remainder of the present specification, the Examiner will recognize that the present invention operates through the GUI, not through an API or other lower-level system call.

Restricting operations to GUI level functions also pertains to detection of objects upon the SUT 290 display. From the present specification on page 11, lines 5 - 9: "Utilizing the above

command set, it is possible to monitor a graphical user interface for any type or shape of image and then, responsive to the presence thereof, select a subsequent user action as though the user action were being performed directly upon the system-under-test 290 rather than from a source or controlling computer."

The use of the SUT 290 GUI as the point of access and interaction therewith has now more clearly been recited by way of the present amendment. No new matter has been added by way of the present amendments. Consequently, in view of the present amendment and remarks, the Examiner is respectfully requested to reconsider the rejection of record and allow the present application to issue. No new matter is introduced. However, should there remain any open issues in this application which might be resolved by telephone, the Examiner is respectfully requested to call the undersigned at 320-363-7296 to further discuss the advancement of this application.

Sincerely,

Albert W. Watkins

reg. 31,676